AI Student Advisor System Design

Overview

This lecture discussed the architecture and workflow of an AI-powered student advisor application, focusing on system design, data flow, and practical development considerations.

Application Workflow & Architecture

- User interacts with a web or mobile app, which sends requests to a web server.
- The initial screen allows students to upload a PDF containing class information.
- The web server forwards the PDF to a middle tier, which processes the request.
- The middle tier sends the PDF and a crafted prompt to an API that communicates with an LLM (Large Language Model), e.g., OpenAI.
- LLM returns structured class data as a JSON object, which includes classes, homework, exam info, and due dates.
- Iterative prompt engineering is needed to optimize JSON extraction from the LLM.
- JSON data is stored in a database for future queries, avoiding repeated LLM calls.

Data Handling & Query Strategies

- The system stores a JSON object per class per student per semester.
- For common queries (e.g., exam dates), the middle tier retrieves and parses JSON directly from the database.
- Complex or natural-language questions can be processed by sending the JSON and the question to the LLM.
- The middle tier acts as the central logic layer, connecting frontend, backend APIs, and the database.

Technical Considerations

- The database choice (e.g., MongoDB vs. Postgres) is flexible and may

evolve based on team needs.
- Middle tier connects to the database directly via a connection string (not via URL endpoints like APIs).
- Building small, focused API components aligns with modern software architecture practices.
- Understanding basic concepts (API, web server, HTTP, database types, JSON) is stressed for effective development.

Project Management & Learning

- Mistakes and experimentation are encouraged as part of the learning process.
- Teamwork, documentation, and version control (e.g., GitHub) are vital for project success and for presenting experience in interviews.
- Students are urged to dedicate consistent time (~1-2 hours/day) and aim for a working prototype by semester's end.

Key Terms & Definitions

- **Web Server** — A system that responds to client requests with web pages or data.
- **Middle Tier** — The logic layer that handles processing and coordination between frontend, backend APIs, and the database.
- **API (Application Programming Interface)** — A set of rules for interacting with software components or services.
- **LLM (Large Language Model)** — An AI model (e.g., OpenAI) that processes and generates human-like language.
- **JSON (JavaScript Object Notation)** — A lightweight data format for structuring information.
- **Connection String** — A configuration that specifies how to connect to a database.
- **Structured Database** — Databases with fixed schemas (e.g., SQL Server, Postgres).
- **Unstructured Database** — Flexible-schema databases (e.g., MongoDB).

Action Items / Next Steps

- Research and understand APIs, JSON, databases (structured/unstructured), and web server basics.
- Decide on database technology (MongoDB or Postgres) based on team needs.

- Develop and test prompt engineering for optimal JSON extraction from PDFs.
- Begin work on application prototype and commit to regular development sessions.

Certainly! Here's an expanded version of your notes with more detailed explanations and context:

Expanded Notes on AI-Powered Student Advisor Application

Overview

This lecture focused on designing and building an AI-powered student advisor application. The main goal is to help students manage their academic information by uploading course documents (like PDFs) and extracting structured data to answer their questions intelligently. The discussion covered system architecture, data flow, technical choices, and project management strategies.

Application Workflow & Architecture

User Interaction

- The user accesses the application via a **web app** or **mobile app**.
- The app presents an intuitive interface, such as a screen where students can upload their course PDFs containing schedules, homework, exams, and due dates.

Request Flow

- When a student uploads a PDF, the app sends this file to a **web server**.
- The web server acts as the entry point, receiving requests and forwarding them to the **middle tier** for processing.

Middle Tier Responsibilities

- The middle tier is the core logic layer that:
  - Receives the PDF and a **prompt** designed to instruct the AI on how to

extract data.
- Sends the PDF and prompt to an **API** that communicates with a **Large Language Model (LLM)**, such as OpenAI's API.
- Receives a structured **JSON** response containing parsed class information.
- Stores this JSON in a database for efficient future access.
- Handles queries from the frontend by either retrieving data from the database or, for complex questions, sending the JSON and query back to the LLM for interpretation.

Prompt Engineering

- Crafting the right prompt is crucial to get accurate and well-structured JSON output from the LLM.
- This is an iterative process: initial prompts may produce imperfect JSON, which is refined over time.
- The JSON structure typically includes:
  - Outer objects representing classes (e.g., "CS2340").
  - Nested objects for homework assignments, exams, and other relevant details.
  - Fields like due dates, chapters covered, professor info, etc.
- Missing data fields are handled gracefully, with placeholders or null values.

Data Handling & Query Strategies

JSON Storage

- Each class's data is stored as a JSON object in the database.
- For a student taking multiple courses, multiple JSON objects exist, one per class per semester.

Query Handling

- Simple queries (e.g., "When is exam 1?") can be answered by scanning the stored JSON directly, avoiding unnecessary calls to the LLM.
- Complex or natural language queries are handled by sending both the stored JSON and the user's question to the LLM, which acts as an intelligent agent to interpret and respond.

Middle Tier as Coordinator

- The middle tier manages communication between:
  - The frontend (web/mobile app).
  - The LLM API.
  - The database.
- It decides when to query the database or the LLM based on the complexity of the request.

Technical Considerations

Database Choices

- The team is considering **MongoDB** (a NoSQL, unstructured database) and **Postgres** (a relational, structured database).
- MongoDB offers flexibility with JSON-like documents, which aligns well with storing JSON objects.
- Postgres provides strong schema enforcement and relational capabilities.
- The choice depends on project needs, team familiarity, and performance considerations.

Connecting to the Database

- The middle tier connects directly to the database using a **connection string**.
- Unlike APIs, databases do not typically expose URL endpoints; instead, they use connection protocols specific to the database type.
- For example, MongoDB uses a connection URI with server address, port, and credentials.
- Postgres uses a similar connection string format.

API Design

- The architecture favors small, focused APIs that do one thing well.
- The API communicating with the LLM is dedicated solely to that purpose.
- The middle tier manages multiple backends (LLM API and database) and abstracts this complexity from the frontend.

Understanding Core Concepts

- Developers should understand:
  - **API**: Interface for software components to communicate.
  - **Web Server**: Handles HTTP requests and serves web pages or data.
  - **HTTP Protocol**: The foundation of web communication.

- **JSON**: Lightweight data format for structured information.
- **Structured vs. Unstructured Databases**: Differences in schema enforcement and flexibility.

Project Management & Learning

Learning Through Experimentation

- Building this application is an iterative learning process.
- Mistakes and redesigns are expected and valuable.
- Experimenting with prompt design and technology choices helps deepen understanding.

Teamwork & Documentation

- Using version control (e.g., GitHub) to manage code collaboratively.
- Documenting design decisions, such as why MongoDB or Postgres was chosen.
- Practicing agile methodologies like Scrum with regular meetings and workload distribution.

Time Commitment & Goals

- Students are encouraged to dedicate 1-2 hours daily to the project.
- Aim to have a working prototype by the end of the semester.
- The project can serve as a portfolio piece for job interviews, demonstrating practical skills and decision-making.

Key Terms & Definitions

TermDefinition**Web Server**Software that handles HTTP requests and serves web pages or data to clients.**Middle Tier**The application layer that processes logic, coordinates between frontend, backend, and DB.**API**A set of rules and protocols for building and interacting with software applications.**LLM (Large Language Model)**AI models capable of understanding and generating human-like text (e.g., OpenAI GPT).**JSON**A lightweight, text-based format for representing structured data.**Connection String**A string containing information needed to connect to a database (server, port, credentials).**Structured Database**Databases with fixed schemas, such as SQL Server or Postgres.**Unstructured Database**Flexible-schema databases like MongoDB that store data in document-like

formats.## Action Items / Next Steps

1. **Research Core Concepts**Study APIs, JSON, HTTP protocol, and database types to build foundational knowledge.
2. **Decide on Database Technology**Evaluate MongoDB vs. Postgres based on project requirements and team skills.
3. **Develop Prompt Engineering**Experiment with prompts to optimize JSON extraction from PDFs using the LLM.
4. **Build Prototype**Start coding the application, focusing on the upload flow, LLM integration, and JSON storage.
5. **Commit to Regular Development**Schedule consistent work sessions to maintain progress and meet semester goals.

If you'd like, I can also help you create a study guide or summarize specific sections further! Just let me know.